第19回（最終回）

形式的検証技術の現状と今後の動向

藤田昌宏

この連載では，システムLSIを中心としたハードウェア設計検証を形式的に行う手法について解説してきました．連載の最後に，形式的検証技術の現状と今後の動向について，いくつかの観点から，個人的視点も含めて説明します．（筆者）

論理式の計算機上における取り扱いに関する技術は，近年大きく進歩しています．そしてそれに関連して，形式的検証技術が適用できる範囲が広がりつつあります．ここではその状況，ならびに仕様記述法の動向などを中心に解説します．

現状をごく簡単にまとめると，形式的検証ツールや手法は，誰でも使えるという段階ではありません．依然として「よく分かっている人が賢く使えば，非常に強力なツールである」という事実は変化していません．しかし，技術の進歩は確実に進んでいること，そして仕様記述言語の標準化が大きく進んだこと，それにツールの使い方に関する解説も増えていることから，以前より，賢く使っている人が増えていることは事実です．いくつかの技術的分野で大きな進歩もあり，システム・レベル設計など，高位設計支援技術の導入が進むであろう今後が楽しみであるといえます．以下では，いくつかの観点から現状と今後について解説します．

与えられた和積形論理式（CNF formula）f：

- 変数の集合 V　　$(a,\ b,\ c)$
- 項（clause）の積　　$(C_1,\ C_2,\ C_3)$
- 項：リテラル（literals；変数かその否定）の和

すべての項の少なくとも一つのリテラルを１とするような変数値の組み合わせが存在するか？

例 ： $\underbrace{(a+b+\overline{c})}_{C_1}\underbrace{(\overline{a}+c)}_{C_2}\underbrace{(a+\overline{b}+c)}_{C_3}$　　$a=b=c=1$

図1　SAT（論理関数の充足可能性判定）問題

通常は積和形（conjunctive normal form ： CNF）で表現された論理式全体を1とするような変数の値の組み合わせが存在するか否かを判定する問題．

1　SAT手法は近年，大きく進歩している

形式的検証に関する基盤技術の中で，近年もっとも研究が進み，結果としてツールの性能が飛躍的に向上しているものに，論理関数の充足可能性（satisfiability）判定手法，いわゆるSAT手法があります．

● SAT問題を解くSATソルバ

SAT問題（論理関数の充足可能性判定問題）とは，**図1**に示すように，通常は積和形（conjunctive normal form ： CNF）で表現された論理式全体を1とするような変数の値の組み合わせが存在するか否かを判定する問題です．問題の難しさの観点からは，NP完全（NP-complete）になります．つまり，論理変数の数に対して，最悪の場合，処理に指数的な時間が必要となると考えられています．

形式的検証手法の多くの問題は，いくつかの変換を行うことにより，SAT問題に変換できることが分かっています．このため，SAT問題と解くツールであるSATソルバ

KeyWord　形式的検証，SAT手法，モデル・チェッキング，Bounded解析，等価性検証，仕様記述手法，PSL，SVA，検証IP

は形式的検証手法にとって，非常に重要な要素技術となっています．

● SAT手法で扱える変数の数が急激に増大

SAT問題に対する基本的なアルゴリズムとしては，DPLLアルゴリズムが1960年代に考案されました．そしてそれ以降，さまざまな改良がなされてきましたが，1990年代までは，扱える論理変数の数はそれほど大きくならず，結果として形式的検証分野からの興味は，それほど大きくありませんでした．

しかし，**図2**に示すように，2000年辺りを境として，扱える変数の数が急激に増大しました．図では，横軸が年代であり，縦軸は扱える変数の数を示しています．問題の難しさは，扱える変数の数に対して指数的に増大するはずです．にもかかわらず，**図2**のグラフでは，年代に対して扱える変数の数があたかも指数的に増大しているのではないかと思われるくらいになっている点に注意する必要があります．これは，計算機科学的な常識ではありえない事が起こっていることになります．

実際には，以下で示すような改良が1990年代後半から行われ始めたためです．これが現在まで続いている結果として，**図2**に示すような規模のSAT問題を解くことのできるSATソルバが開発されています．

- 基本アルゴリズムは変数の場合分けを繰り返し，途中で矛盾が生じたらほかの場合を総当たり的に調べるということであるが，一度矛盾が生じた場合にその原因を特定し，同じような矛盾に対する解析を二度としないように改良した．
- 変数の値が部分的に決まった際の影響がほかにも伝播することを計算するアルゴリズムを大きく改良した．
- 実際のプログラムとして性能向上を図るために，キャッシュ・ヒット率なども考慮してデータ構造や処理アルゴリズムをチューニングした．
- 変数の場合分け順の決定アルゴリズムにさまざまな工夫を施した．

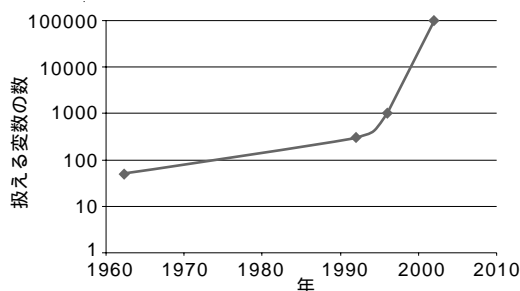



図2　SAT手法が急速に進歩
2000年辺りを境として，扱える変数の数が急激に増大した．

SATソルバに対するコンテストが毎年開催されています．このコンテストでは，前年度の優勝ツールそのままでは次年度に入賞すらできません．SATソルバについては，このような速いペースの性能改善が今でも続いています．結果として，現状で，数百万変数からなる論理式の充足可能性判定を数時間で行えるまでになっています．

論理式を計算機で扱うための技術として，2分決定グラフ（BDD：binary decision diagram）があります．しかしBDDでは数百変数の論理式を扱うのが限度です．このことを考えると，SATソルバの性能は驚異的であると言えます．

● SATソルバでは10万ゲート程度まで扱える

順序回路の検証の場合，問題の複雑さという観点からは，回路中のゲート数と順序回路を時間軸方向に何サイクル解析するかということから議論できます．

今，10万ゲートの回路があったとします．それを10サイクル解析する問題をSAT問題に変換すると，（10万×10サイクル×各ゲートの入出力数）だけ変数が現れ，数百万変数が必要になります．現状のSATソルバの性能はそれくらいが限界であるため，その意味で，10万ゲート程度まで扱えることになります．

この規模は，設計者が自分の設計部分を気軽に検証するには少し小さいかもしれません．しかし，設計をいくつかに分ければ，検証できることを意味しています．今後もSATソルバの進歩は続くと考えられることを考慮すると，SAT手法による形式的検証は実用段階に入っていると言えるでしょう．

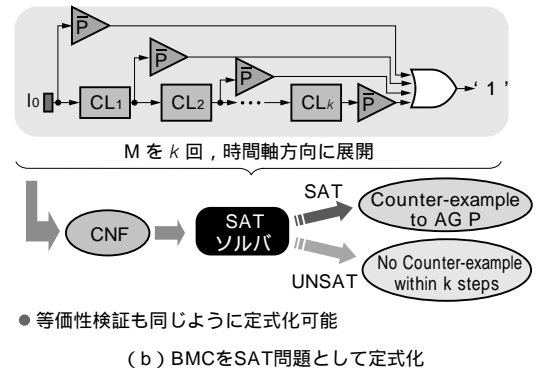
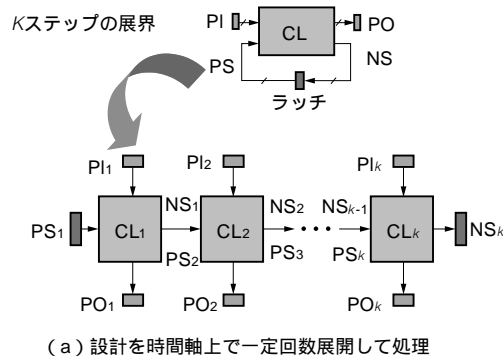
商用のモデル・チェッキング・ツールなどでも，SATソルバが内蔵されているのが普通です．そして，最近の進歩がそのまま取り込まれることも多くなっています．

また，高位設計検証では，扱う変数が整数型など論理変数に分解したくない場合があります．そのような整数変数を含む論理式のある範囲のものは，一定の自動変換によって，SAT問題に帰着する場合も多くあります．高位設計検証ツールでは，そのような変換も利用しながら，SATソルバが効果的に利用されています．

なお，最新のソルバも含めて，多くのSATソルバは，パ

(a) 設計を時間軸上で一定回数展開して処理

(b) BMCをSAT問題として定式化

図3
Boundedモデル・チェッキング手法
(a)のように，順序回路を設計者が指定した回数だけ時間軸上に展開した回路を検証対象とする．

ブリック・ドメインのツールとして，Webからダウンロードできるものがたくさんあります．そこで，扱う問題に合わせて自分でSATソルバを改良して利用することを行っているグループも数多くあります．非常に高度な使い方ですが，自分で検証ツールを改良しながら使うことができる環境では，価値は非常に高くなります．

2 モデル・チェッキング技術の現状と動向

モデル・チェッキングの基本アルゴリズムは，1980年代初めに提案されました．その後さまざまな改良が加えられ，現在ではハードウェア設計用モデル・チェッキング・ツールとして，数社から提供されています．それらのツールは細部はかなり異なると考えられますが，基本的なアルゴリズムは共通部分が多いといえます．

近年のSATソルバの性能向上の結果がそのまま現れる形で，モデル・チェッキング・ツールが扱える設計規模も増大しています．10万ゲート規模の設計が検証可能となっており，将来的にはより大規模な回路の検証も行えると考えられます．

● 順序回路は指定した回数だけ展開する

ここで少し注意すべきこととして，モデル・チェッキングの際，設計を何サイクルの範囲で解析する必要があるかという点があります．

設計を網羅的にカバーするには，すべての状態遷移がループになるまで回路を解析し続ける(このことを一般にfixed point計算と呼ぶ)必要があります．しかしこれでは，回路によっては非常に多数のサイクルに渡って解析しなければならないことになります．

一般にSATソルバを利用する場合には，数学的帰納法を利用してfixed point計算と等価なことを行うのも技術的には可能です．しかしこの場合，扱える回路規模が小さくなってしまうという問題があります．

そこで現状では，**図3**に示すように，順序回路を設計者が指定した回数だけ時間軸上に展開した回路を検証対象とするBoundedモデル・チェッキング手法が広く利用されています．このように順序回路を時間軸上で展開すれば，結果としては，元の順序回路中の組み合わせ回路が展開した回数だけ増大した組み合わせ回路となります．従って，そのモデル・チェッキングはSAT問題としてそのまま定式化できます．結果として高性能SATソルバを適用し，比較的大規模設計に対するモデル・チェッキングが行えることになります．

逆に言うと，展開した回数を超えるような解析は一切していないわけです．もし仕様に対する反例がその回数を超えるものしか存在しない場合には，決して見つからないことになります．

● 設計を抽象化する手法の研究が進む

そこで，まず与えられた設計の抽象化を行い，その抽象化されたものに対して通常のモデル・チェッキングを行う手法についての研究が進められています．これは一部の商用ツールにも導入されつつあります．

ここで言う設計の抽象化とは，最も単純には，元の設計の複数の状態を一つにまとめる処理のことです．抽象化された設計の状態数が小さくなるため，抽象化をどんどん進めれば，必ずいつかはモデル・チェッキングが可能となります．

しかし，単純に状態をまとめただけでは，嘘の反例が生成されることがあります．これは，抽象化された設計は正しくないが，元の設計は正しい場合に相当し，抽象化された設計に対する反例に対応する実行シーケンスが元の設計にないことを意味します．このような場合には，その反例が反例ではなくなるように設計抽象化処理の改良が行われます．

この改良を自動的に行う手法に関する研究が現在活発に行われています．また，その中でもSAT手法が利用されています．つまり，SATソルバの改良に伴い，設計抽象化の自動改良手法の性能の向上も期待できるわけです．

● ソフトウェアに対する手法のハードへの適用に期待

ソフトウェアに対するモデル・チェッキング手法の研究も進んでいます．

JAXA言語やC系言語に対するモデル・チェッカが研究ツールとして公開されています．また，実用規模のソフトウェアの検証を行った例も報告されるようになっています．

例えば，Linuxのカーネル・コード（400万行程度）のモデル・チェッキングを行い，実際にいつくかのバグを発見したという報告があります．これは，上で述べたプログラム記述の抽象化を施した上で，SATソルバで検証したものです．数十台のワークステーションで並列処理を行うことにより，数時間で検証しています．これは，SATソルバの性能を最大限に生かしたものであると言えるでしょう．

ソフトウェア開発では，プログラム・コードをルール・ベース的に判定するLINTツールもよく利用されています．しかし，ルールを詳細化したり，モデル・チェッキングのアイデアを利用することで，より詳細な検証を行う手法についても研究開発が進められています．

ソフトウェアに対する検証ツールは，一般にC言語系が扱えるため，Cベース言語によるハードウェア設計で利用することも考えられます．この種のツールでは，設計記述として好ましくないパターンをルールの形で定義し，そのパターンにヒットするコードを効率良く探すことで検証しています．ルールにはユーザが定義できるものもあり，ハードウェア高位設計用のものも開発可能です．

今後，高位設計支援が進むにつれ，このような考えに基づくツールも利用が進むと考えられます．

3 等価性検証技術の現状と動向

組み合わせ回路同士の等価性検証については，2000年ごろから，既に実用ツールとして広く利用されています．

等価性を比較したい二つの順序回路について，フリップフロップの対応が分かっている場合に相当します．テープアウトする設計と，シミュレーションなどでしっかり検証された回路との比較を最後に行うという使われ方がよくあります．

● 組み合わせ回路の等価性検証はすでに実用レベル

このような組み合わせ回路用の等価性検証ツールでは，比較している二つの回路どうしは「近く」，内部に互いに等価である点が多数存在するという場合には極めて強力です．1000万ゲートを越える設計同士の等価性も数時間で判定できるようになっています．これは，回路全体に対する等価性問題を内部等価点で分割し，分割された各部分回路同士の等価性判定を繰り返すことで解いているからです．回路規模が増大しても内部等価点が多数ある限り，検証時間の増大もある程度抑えられるのです．

一方，もし内部等価点が少数しかない場合には，数十万ゲート規模でも検証できないこともあります．内部回路構造がまったく異なる場合や，RTL（resister transfer level）設計の記述に多数の内部ドント・ケアが存在する場合などが相当します．このような場合には，先ほどのSATソルバを単純に回路全体に適用して等価性を検証するしかなく，扱える回路規模は，基本的にSATソルバの性能ということになります．

このように，組み合わせ回路に対する等価性検証はかなり実用的なツールとなっています．

● 順序回路の等価性検証は比較的小さい設計に限られる

一方，等価性を比較したい順序回路同士で，フリップフロップの対応が付かない場合には，組み合わせ回路の等価性問題に帰着できなくなり，順序回路としての等価性を直接調べる必要があります．

Cベース言語などによる高位設計記述から設計を開始する場合に，上位設計と下位設計の等価性判定を行おうとすると，現在では順序回路用の等価性判定ツールが必要になります．また，RTLから設計を開始する場合でも，回路中

の組み合わせ回路を部分的にフリップフロップをまたいで移動させることで，回路全体のタイミング調整を行うリタイミングと呼ばれる手法を利用している場合には，やはり組み合わせ回路用の等価性判定ツールでは扱えなくなり，順序回路用の等価性判定ツールを利用することになります．

順序回路用の等価性検証でも，モデル・チェッキングと同じように，初期状態からすべての状態を網羅するまで状態遷移を解析するfixed point解析と，一定サイクル数しか解析しないBounded解析に分けることができます．fixed point解析が可能となるのは，モデル・チェッキングと同じで，一定規模以下の比較的小さい設計に限られてしまいます．そこで，実用的にはBounded解析が広く利用されています．

なお，等価性検証の場合，二つの設計を比較することになるので，検証ツールが扱わなければならない規模は，2倍になることにも注意が必要です．

図4　途中状態からのBounded解析

通常は初期状態から検証するが，Bounded解析を効率良く行うには，設計者の指定した状態から行う場合も多い．このような場合には，指定した状態が意味のある常態であるか否か（つまり，初期状態から到達可能な状態に含まれるか否か）は設計者の責任で行うことになる．

● 順序回路の等価性の意味に注意

順序回路の等価性検証で，まず注意すべきことは，等価とはどういうことかということです．

組み合わせ回路の場合には，現在の入力が決まれば，出力が決定するため，等価性の定義は自明でした．しかし，順序回路の場合には，入力を受け取るタイミングと，出力が送出されるタイミングは各設計で異なるのが普通です．従って，そのようなタイミングも含めた等価性の定義を設計者が行う必要があります．これは，場合によってはかなり複雑で，等価性の定義に誤りがあるという場合も決してなくはないということを注意する必要があります．

等価性検証であれ，モデル・チェッキングであれ，Bounded解析の場合は，例えば初期状態（通常は，リセット状態）から一定サイクル数だけ，等価性を検証することになります．しかし回路の性質によっては，あまり意味がないこともあります．

例えば，キャッシュ・コントローラを検証することを考えてみます．初期状態では，キャッシュ・メモリは一般に空です．従って，初期状態から100サイクルや1000サイクルについて解析して検証したとしても，キャッシュ・メモリがいっぱいになることはありません．一方，キャッシュ・コントローラの設計におけるバグ（不具合）は，キャッシュがいっぱいになった場合の扱いに関することが多いものです．この意味では，Bounded解析はあまり意味がないことになってしまいます．

● ツールの使い方のノウハウが重要

実際には，リセット状態から回路を一定サイクル解析するのではなく，設計者が指定した状態から一定サイクル解析するような工夫を施して利用するのが普通です．

このような場合には，**図4**に示すように，設計者がどのような状態から解析するように指定するかで意味のある検証ができるかどうかが決まります．結果として，バグを発見できるかが大きく異なってきます．もし実際には起こらないような状態から解析を始めるように指定してしまうと，実際の設計では起こりえないような動作を中心に検証していることになってしまいます．逆に，うまい状態を指定できると，容易にバグを発見できます．

どのような状態を指定すべきかは，各設計に大きく依存します．さらに言うと，利用する検証ツールが採用している検証手法にも依存するため，効率的な状態の指定を行うには，ツールごとの利用経験やトレーニングが必要になる場合もあります．このような，一種の使い方のノウハウに関する部分も重要であることには違いなく，いかに教育していくかも検討する必要があるといえるでしょう．

4 仕様記述手法の現状と動向

以前は，モデル・チェッキング・ツールに与える仕様の表現の仕方は，各ツールでまちまちでした．しかし最近は，

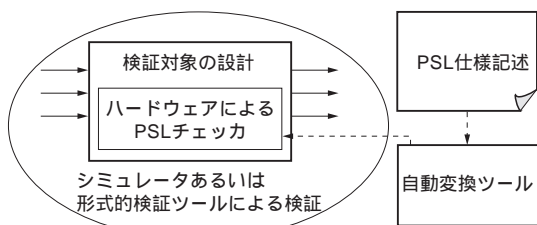



図5　PSLから回路を自動生成して検証
与えられたPSLによる仕様記述を自動的に回路に変換し，その回路と元の設計を同時に処理することで，設計検証を行う．

PSL(property specification language)やSVA(system verilog assertion)などの標準化が進み，ツールに関係なく同じ形で仕様を与えられるようになりつつあります．

● 仕様記述はPSLとSVAが普及

PSLとSVAは，基本的には同じものであると考えられます．

SVAはVerilog HDLを元にした構文になっています．PSLは，同様のことを一般的に定義しています．単なる論理式ではなく，時間シーケンスを表現するための正規表現の導入，動作を表現するための状態遷移記述，さらにモデル・チェッカなどで広く利用されている時相論理(temporal logic)を用いたプロパティも記述できるようになっています．このようにさまざまな工夫がされた言語であるといえますが，言い換えると言語としては非常に大きいものになっています．そのため，言語全体をサポートする検証ツールの開発は必ずしも容易ではありません．

● 仕様記述を回路に変換して検証する研究が進む

PSLで表現された仕様を回路に変換して，シミュレーションや形式的検証手法で検証を行う手法も提案されています．

図5に示すように，与えられたPSLによる仕様記述を自動的に回路に変換し，その回路と元の設計を同時に処理することで，設計検証を行うものです．PSLでは一般的な記述が扱えるため，検証ツールに依存しない検証手法として価値は高いと考えられます．また，この手法を発展させると，PSLの仕様記述から設計を自動生成できる可能性もあります．現在，いくつかの例で動作が確認されている状況であり，今後が期待される研究です．

● 検証IPの流通が始まる

一方，仕様記述言語が標準化されると，仕様記述が商品として流通するようになってきます．例えば，ARMプロセッサのAMBAバスやオンチップ・バスの標準の一つであるOCP(open core protocol)などに関する仕様(プロパティの集合)は，いくつかの検証ベンダからすでに供給されています．

このように，一般に使われているものに対する仕様は，既存のものを検証IP(intellectual property)としてそのまま利用することが可能となっています．モデル・チェッキングの場合，正しいプロパティを作成するというのは必ずしも容易なことではありません．このため，この種の検証IPは今後，さらに広く利用されると考えられます．

一般に流通しない場合でも，一つの設計プロジェクト内で多世代に渡る設計において，検証IPとして蓄積・再利用が進むと考えられます．

＊　＊　＊

以上，形式的検証手法の技術動向について，いくつかの観点から説明しました．今後は，これらのほかにも，高位設計手法との関連における形式的検証技術，ハードウェア，ソフトウェア協調設計に関する形式的検証技術，さらに，ディジタル回路だけでなく，アナログ回路も含めたミックスト・シグナル設計に対する形式的検証手法などの研究が活発に続けられると考えます．

ふじた・まさひろ
東京大学 大規模集積システム設計教育研究センター(VDEC)教授

<筆者プロフィール>
藤田昌宏．1985年，東京大学大学院 情報工学専門課程修了．工学博士．同年，富士通研究所入社．論理設計用CAD技術の研究開発に従事．1993年より米国富士通研究所へ出向，CAD研究グループを立ち上げる．2000年3月より東京大学大学院 工学系研究科電子工学専攻 教授．2004年4月より東京大学 大規模集積システム設計教育研究センター(VDEC)教授．ディジタル・システム設計支援技術，とくにシステム・レベル合成，論理合成，論理検証を中心に研究している．